Introduction to Safety Systems in Research Accelerators

Software

USPAS

June, 2004

Outline

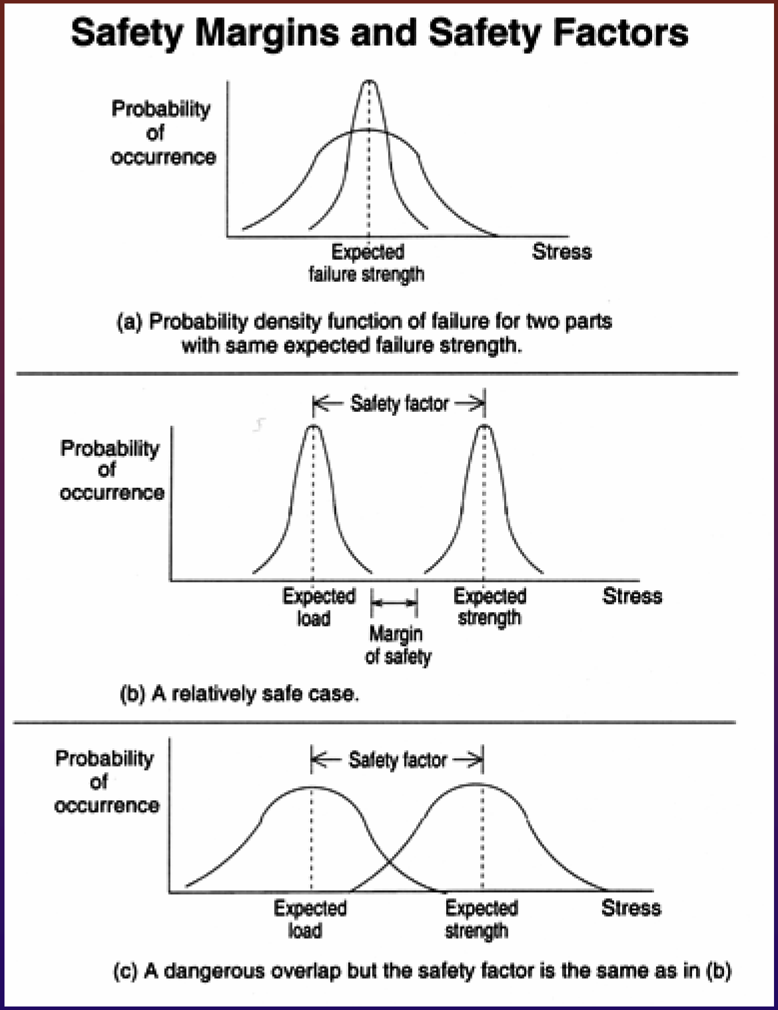
- Overview of software considerations for use in safety applications

- Objective
  - Introduce some of the concerns in using programmable devices and some of the methods used to address them.

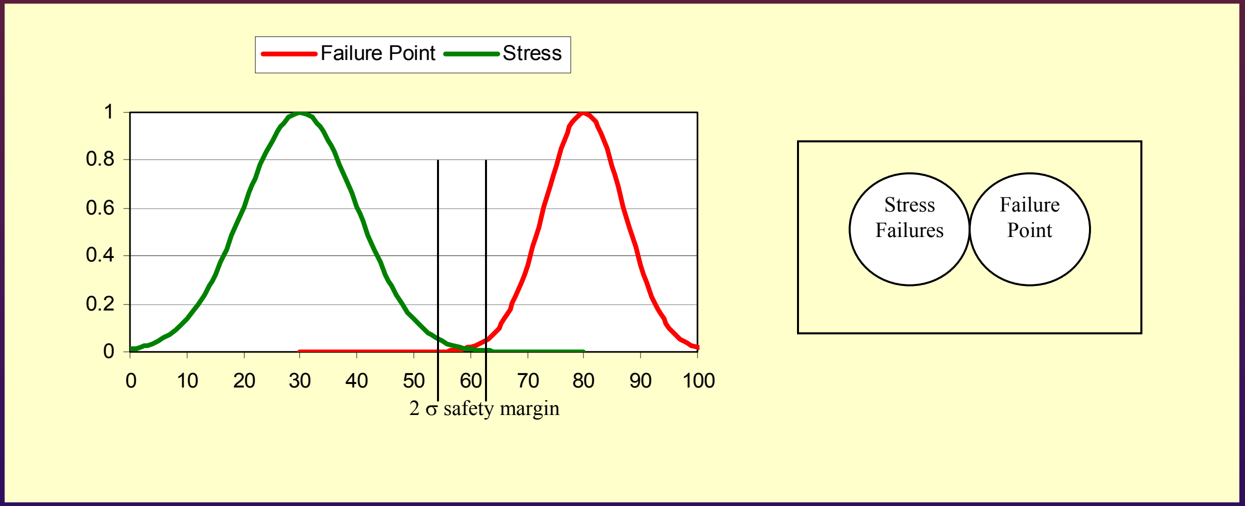
- Nancy Leveson will argue that “software” cannot fail, only hardware. Software is an abstract concept executed by physical hardware.

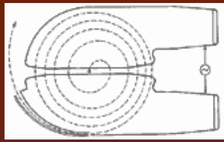
- A stress-strength model can be used.
- Instead of physical stress on a component, software is stressed by demands placed on the constraints within the context of the system.
- These constrains can be:
  - physical, e.g. hardware failure,…
  - logical, e.g. out of bounds data,…
  - temporal, e.g. old data, mis-synchronized functions,…
- It is a matter of how well the constraints are defined and how well the system can handle excursions beyond the constraints.

Stress Strain

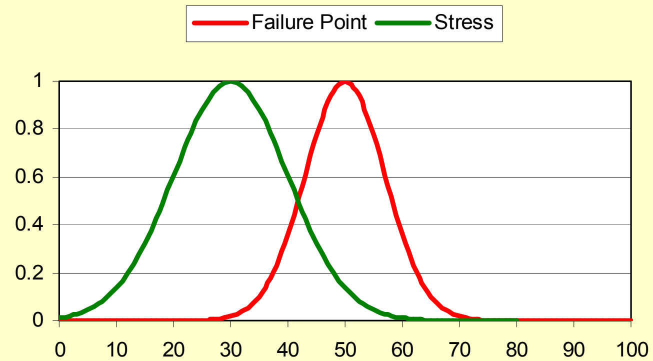


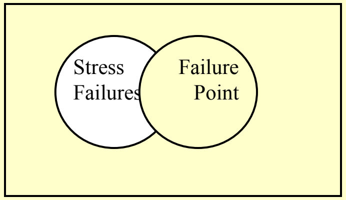
Safety Margin



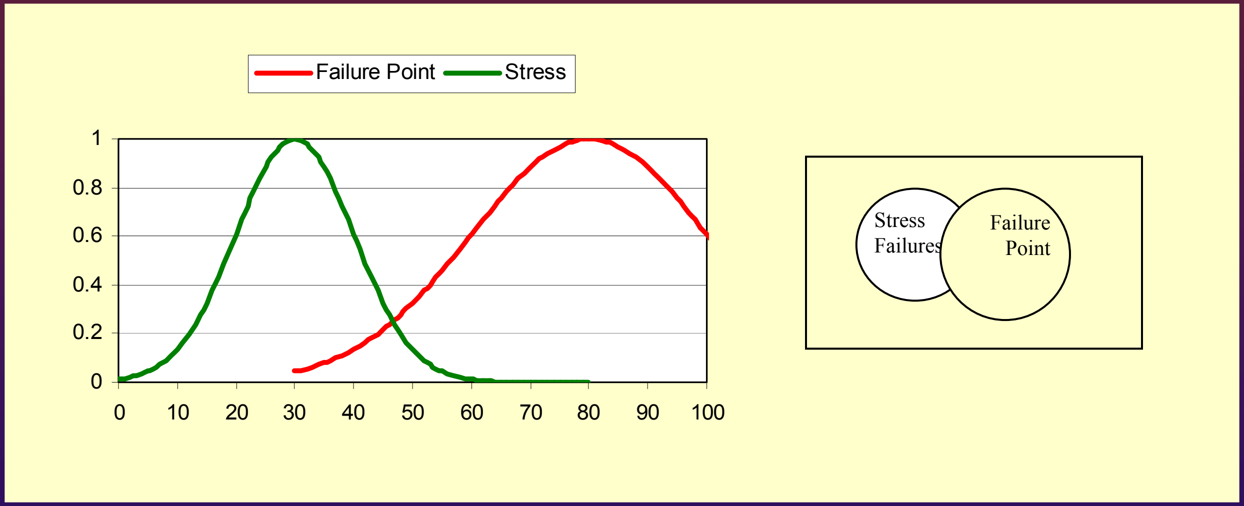


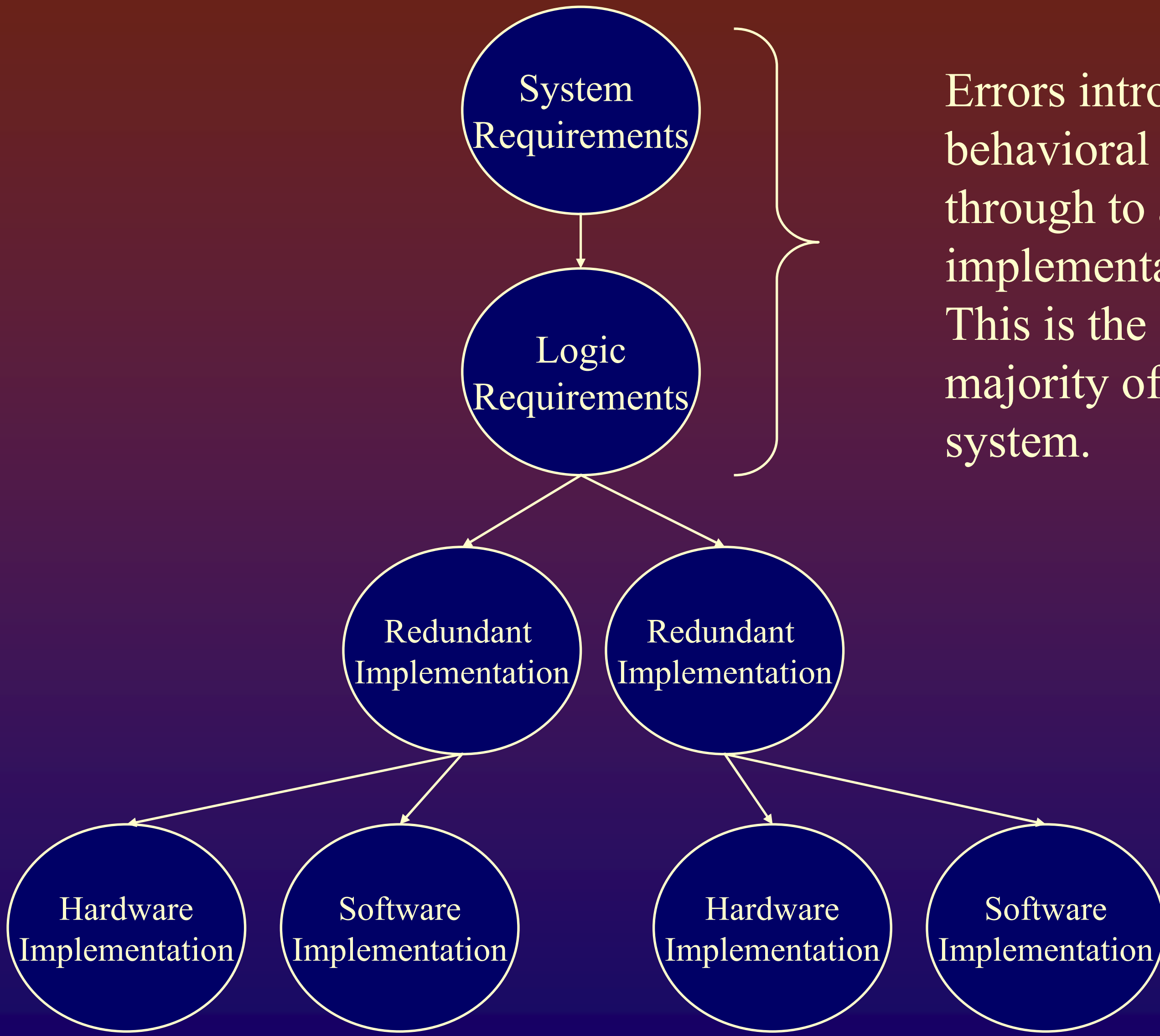
Increase in Failures Due to Insufficient Safety Margin





Increase in Failures Due to Poor QA





Errors introduced in the behavioral phase will propagate through to any type of system implementation.
This is the source of the majority of functional errors in a system.

Requirements

The most important document in safety systems is the requirements document.

Requirements should include

- Context
- Scope and intended use
- Constraints
- Assumptions
- Desired behavior
- Timing requirements
- Exception handling
- Verification/Validation requirements
- Definition of inputs and expected outputs

- Languages
  - IEC61131-3 Defines PLC programming Languages
- Applications
  - Software application development is left to “Good Practice”
  - A good start is in IEC 61508 and 61511
  - IEC880 (Software for Computers in the Safety Systems of Nuclear Power Stations) is a good reference

Programming Languages

- Three Categories
  - Fixed Program Language
    - Application is unalterable
      - Ex. Smart Transmitter
  - Limited Variability Language
    - Well defined functions may be programmed within a structured framework
      - Ex. Ladder Logic, Instruction List, Structured Text
  - Full Variability Language
    - General purpose programming language
      - Ex. ADA, C, C++

Safety Software Design

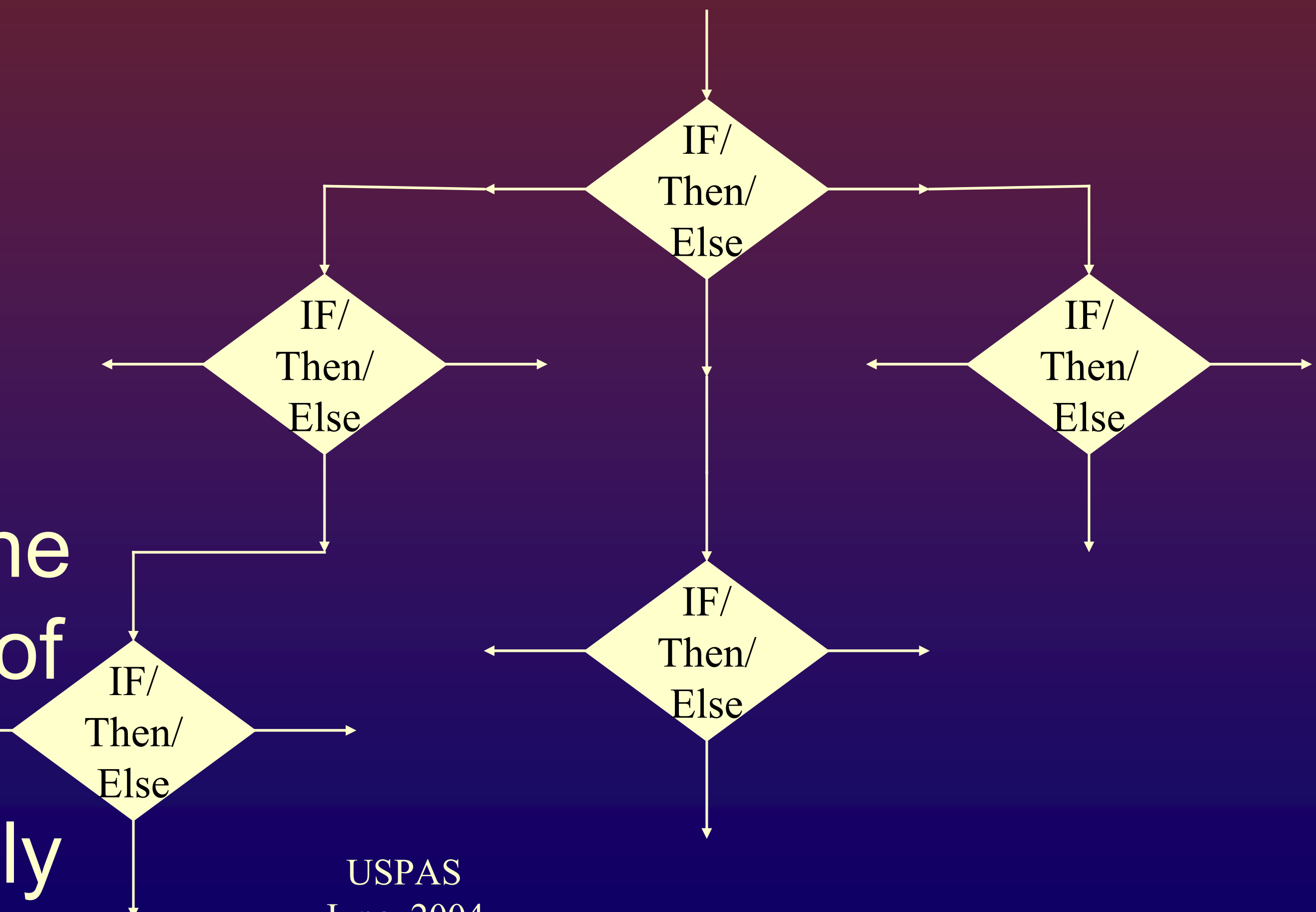
Really, it is high QA design.

Apply standards and good practice that reflect lessons learned from past accidents. Includes things like checklists.

Make use of hazard analysis techniques to help avoid introduction of systematic errors.

Branches

- Every decision branch in a logical system increases the complexity of the system exponentially



Software Analysis Techniques

- Software FMEA
- HAZOP
  - –Hazard and Operability analysis
  - –Qualitative
  - –Carried out on design, not a FMEA
- Fault/Event Trees
  - –Quantitative
  - –Only follows defined faults/events
- Formal Methods
  - –Rigorous but unwieldy

IEC 61508 Part 3 Software

- Defines requirements for software practices based on target SIL level.
- Includes appendices with recommended practice.
  - Practice may be:
  - HR — Highly Recommended
  - R — Recommended
  - --- — mute/no recommendation
  - NR — Not Recommended

Recommendations from IEC 61508 Part 3-Software

Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
1 Use of coding standard		HR	HR	HR	HR
2 No dynamic objects		R	HR	HR	HR
3a No dynamic variables		---	R	HR	HR
3b Online checking of the installation of dynamic variables		---	R	HR	HR
4 Limited use of interrupts		R	R	HR	HR
5 Limited use of pointers		---	R	HR	HR
6 Limited use of recursion		---	R	HR	HR
7 No unconditional jumps in programs in higher level languages		R	HR	HR	HR

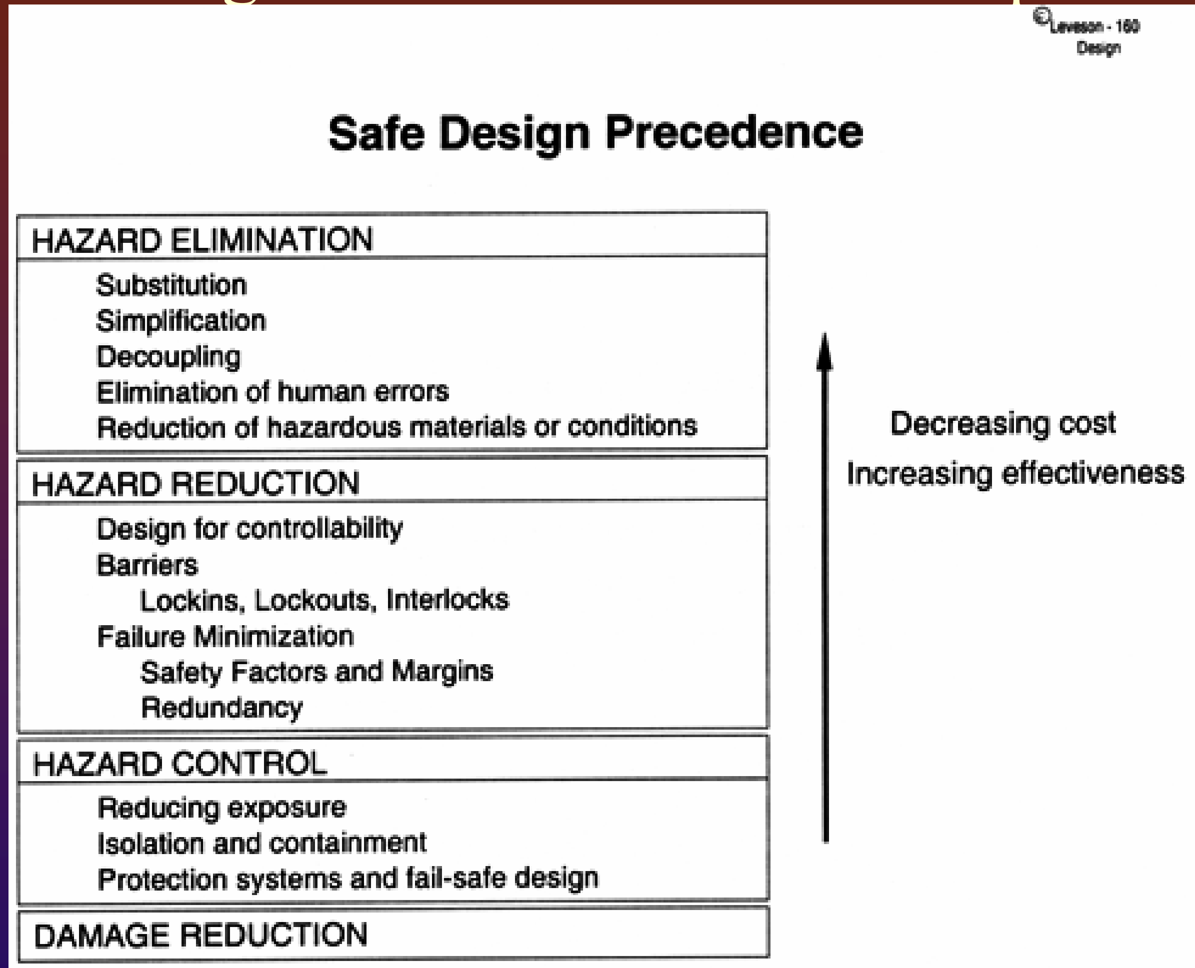
Table B.1 – Design and coding standards

Recommendations from IEC 61508 Part 3-Software

Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
1 Software module size limit		HR	HR	HR	HR
2 Information hiding/encapsulation		R	HR	HR	HR
3 Parameter number limit		R	R	R	R
4 One entry/one exit point in subroutines and functions		HR	HR	HR	HR
5 Fully defined interface		HR	HR	HR	HR

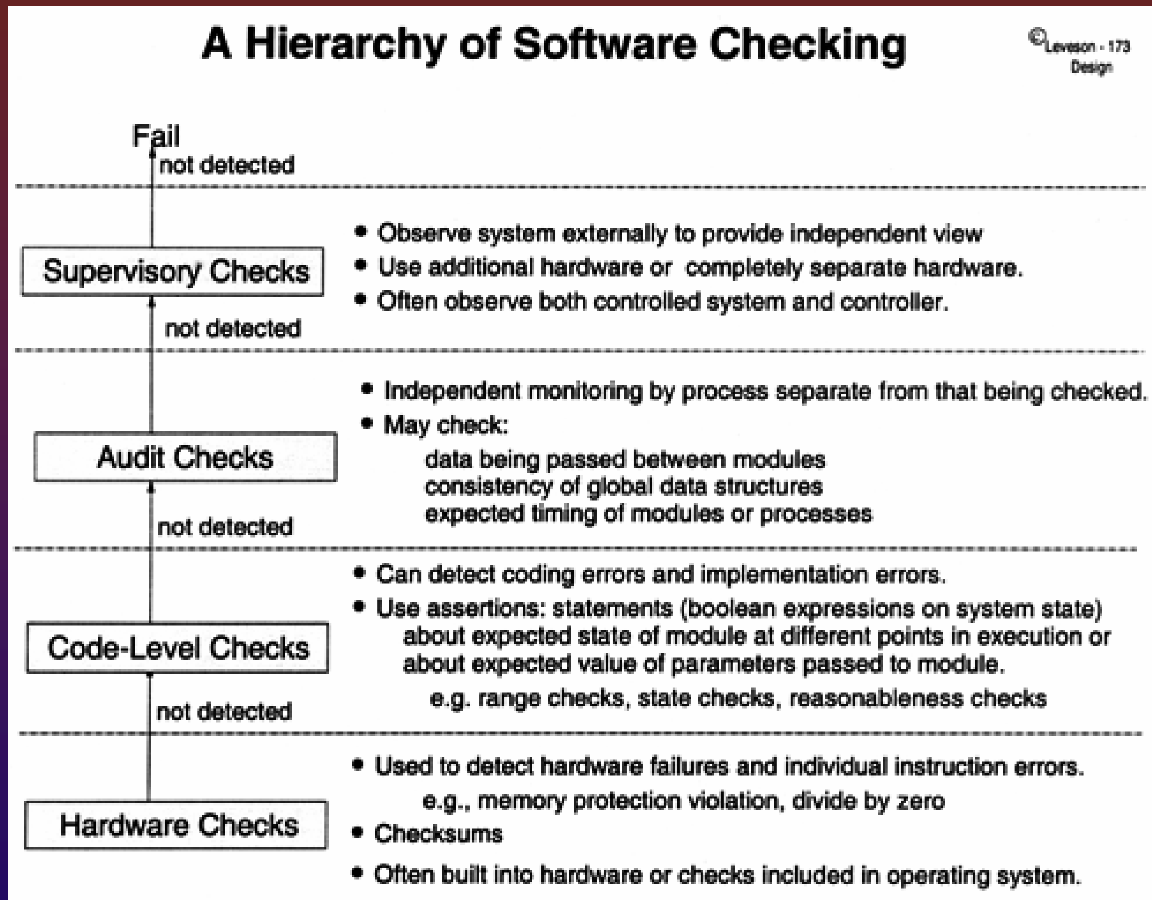
From Table B.9 – Modular approach

Hazard Mitigation from Software Perspective



N. Leveson

Software Checking



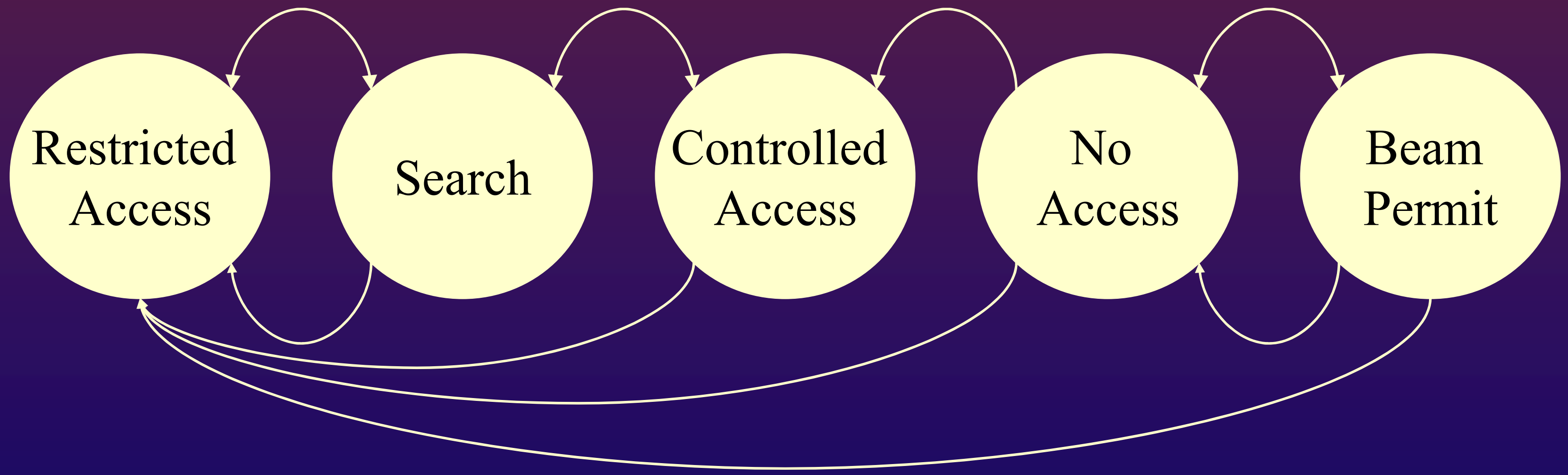
N. Leveson

Self-Checking Software (2)

	Already Known Errors				Other Errors Detected			Added Errors
		Detected						
	#	SP	CR	CD	SP	CR	CD	
3a			1					
3b	4							
3c								
6a			2				1	1
6b	3							1
6c								
8a				2				
8b	2							1
8c							1	3
12a		1					1	
12b	2							2
12c			1				1	2
14a								
14b	2							4
14c								
20a								1
20b	2		1		1			2
20c			1					
23a		2						4
23b	2							
23c								
25a			2				1	
25b	3			1				1
25c								
Total	60	3	8	3	1	0	5	22
		Spec	Read	Chks	Spec	Read	Chks	
			KNOWN			NEWLY FOUND		ADDED

State Machine Design

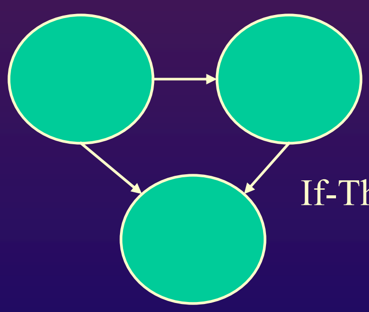
- State or state machine based design
  - Each state must be complete
  - Each state and transition in-to and out-of must be deterministic, e.g. fail safe states.
  - Define “safe” states and “dangerous” states
- Error handling for each condition/state/transition



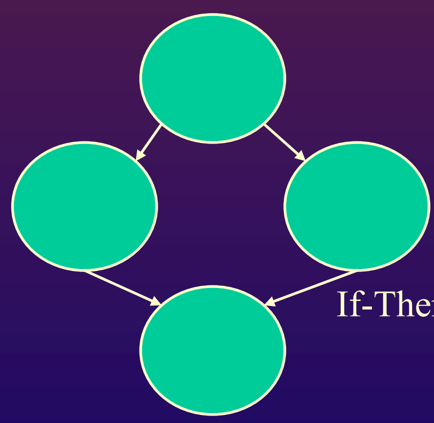
McCabe Complexity

- e is number of edges
- n is number of states

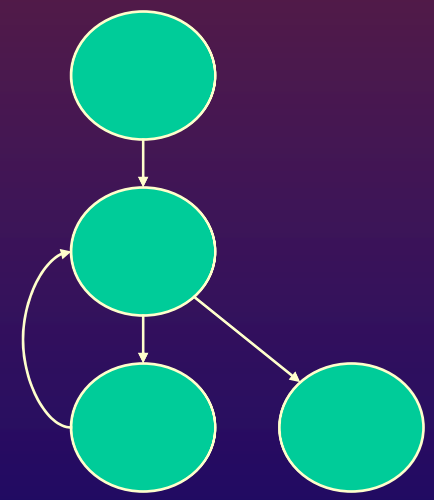
$$Paths = e - n + 2$$



If-Then



If-Then-Else



While-Do